## 一、实验原理

* 1. **netty概述**

Netty 是一个利用 Java 的高级网络的能力，隐藏其背后的复杂性而提供一个易于使用的 API 的客户端/服务器框架。

Netty是基于NIO（Nonblocking I/O，非阻塞IO）开发的，对比于BIO（Blocking I/O，阻塞IO），它的并发性能得到了很大提高。另外，Netty修复了已经发现的所有JDK NIO Bug（比如臭名昭著的epoll Bug），开发人员不需要再为之烦恼。

* 1. **Http协议概述**

要通过 netty实现 HTTP 服务端（或者客户端），首先得了解 HTTP 协议。

HTTP 协议是请求/响应式的协议，客户端需要发送一个请求，服务器才会返回响应内容。例如在浏览器上输入一个网址按下 Enter，或者提交一个 Form 表单，浏览器就会发送一个请求到服务器，而打开的网页的内容，就是服务器返回的响应。

下面讲下 HTTP 请求和响应包含的内容。

HTTP 请求有很多种 method，最常用的就是 GET 和 POST，每种 method 的请求之间会有细微的区别。下面分别分析一下 GET 和 POST 请求。

* + 1. **GET请求**

GET是最常用的HTTP方法。通常用于请求服务器发送某个资源。指定的资源经服务器端解析后返回响应内容。也就是说，如果请求的资源是文本，那就保持原样返回；如果是像CGI那样的程序，则返回经过执行后的输出结果。GET请求报文包含请求行和首部字段。

* + 1. **POST请求**

POST方法用来传输实体的主体。POST请求报文包含三个部分：请求行、首部字段、报文主体，比GET请求多了一个报文主体。首部与报文主体之间用一个空行分割。POST 请求的参数不在 URL 中，而是在报文主体中，首部中多了一项 Content-Length 用于表示报文主体的字节数，这样服务器才能知道请求是否发送结束。这也就是 GET 请求和 POST 请求的主要区别。

* + 1. **HTTP POST Content-Type**

HTTP/1.1 协议规定的 HTTP 请求方法有 OPTIONS、GET、HEAD、POST、PUT、DELETE、TRACE、CONNECT 这几种。其中 POST 一般用来向服务端提交数据，本文讨论主要的几种 POST 提交数据方式。

HTTP 协议是以 ASCII 码传输，建立在 TCP/IP 协议之上的应用层规范。规范把 HTTP 请求分为三个部分：状态行、请求头、消息主体。类似于下面这样：

<method> <request-URL> <version>

<headers>

<entity-body>

协议规定 POST 提交的数据必须放在消息主体（entity-body）中，但协议并没有规定数据必须使用什么编码方式。实际上，开发者完全可以自己决定消息主体的格式，只要最后发送的 HTTP 请求满足上面的格式就可以。

但是，数据发送出去，还要服务端解析成功才有意义。一般服务端语言如 php、python 等，以及它们的服务端框架，都内置了自动解析常见数据格式的功能。服务端通常是根据请求头（headers）中的 Content-Type 字段来获知请求中的消息主体是用何种方式编码，再对主体进行解析。POST 提交数据方案，包含了 Content-Type 和消息主体编码方式 Charset 两部分。以下是几种主要的Content-Type。

* + - 1. **application/x-www-form-urlencoded**

这应该是最常见的 POST 提交数据的方式。浏览器的原生 Form 表单，如果不设置 enctype 属性，那么最终就会以 application/x-www-form-urlencoded 方式提交数据。请求类似于下面这样（无关的请求头在本文中都省略掉了）：

POST http://www.example.com HTTP/1.1

Content-Type: application/x-www-form-urlencoded; charset=utf-8

title=test&sub%5B%5D=1&sub%5B%5D=2&sub%5B%5D=3

首先，Content-Type 被指定为 application/x-www-form-urlencoded；其次，提交的数据按照 key1=val1&key2=val2 的方式进行编码，key 和 val 都进行了 URL 转码。大部分服务端语言都对这种方式有很好的支持。

很多时候，我们用 Ajax 提交数据时，也是使用这种方式。例如 JQuery 的 Ajax，Content-Type 默认值都是 application/x-www-form-urlencoded; charset=utf-8。

* + - 1. **multipart/form-data**

这也是一个常见的 POST 数据提交的方式。使用表单上传文件时，必须让 Form 的 enctyped 等于这个值。

这种方式一般用来上传文件，各大服务端语言对它也有着良好的支持。

上面提到的这两种 POST 数据的方式，都是浏览器原生支持的，而且现阶段原生 Form 表单也只支持这两种方式。但是随着越来越多的 Web 站点，尤其是 WebApp，全部使用 Ajax 进行数据交互之后，可以定义新的数据提交方式，给开发带来更多便利。

application/json 这个 Content-Type 作为响应头大家肯定不陌生。实际上，现在越来越多的人把它作为请求头，用来告诉服务端消息主体是序列化后的 JSON 字符串。由于 JSON 规范的流行，除了低版本 IE 之外的各大浏览器都原生支持 JSON.stringify，服务端语言也都有处理 JSON 的函数，使用 JSON 不会遇上什么麻烦。

JSON 格式支持比键值对复杂得多的结构化数据，这一点也很有用，当需要提交的数据层次非常深，就可以考虑把数据 JSON 序列化之后再提交。

这种方案，可以方便的提交复杂的结构化数据，特别适合 RESTful 的接口。各大抓包工具如 Chrome 自带的开发者工具、Fiddler，都会以树形结构展示 JSON 数据，非常友好。

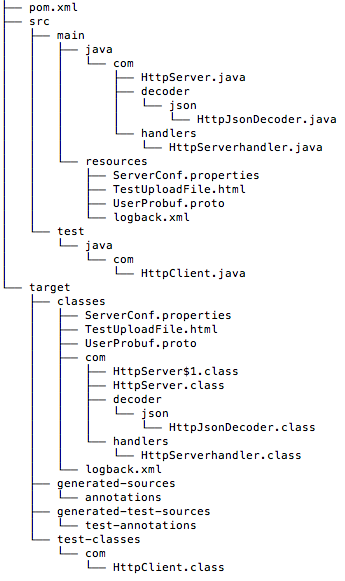
* 1. **netty HTTP编解码**

netty5 提供了对 HTTP 协议的几种编解码器：HttpRequestDecoder、HttpResponseEncoder、HttpServerCodec，分别实现数据的解码、编码以及两者的结合。然而，以上编解码器只能够支持部分 HTTP 请求解析，比如 HTTP GET请求所传递的参数是包含在 uri 中的，因此通过 HttpRequest 既能解析出请求参数。但是，对于 HTTP POST 请求，参数信息是放在消息体中的，所以以上编解码器并不能完全解析 HTTP POST请求。

要实现对POST请求的处理，可以通过HttpObjectAggregator把HttpMessage和HttpContent聚合成一个FullHttpRequest或者FullHttpResponse对象。

## 项目结构

**2.1 项目文件结构**



**2.2程序功能流程图**

程序整体功能如图2.1所示。本实验实现了HTTP协议的部分功能：支持GET方法发送请求、POST方法发送Content-Type为1.2.3.1 application/x-www-form-urlencoded或application/json请求。

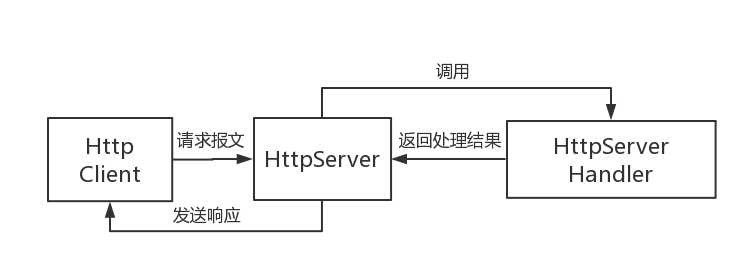


图2.1 程序整体功能

在Netty中，服务器端请求的处理由其绑定的IO事件的处理类完成，自定义的处理类要实现ChannelHandler接口，或者继承ChannelHandlerAdapter抽象类。本实验中自定义 HttpServerHandler 处理类实现处理逻辑。ChannelHandler接口的ChannelRead 函数负责处理请求，ChannelReadComplete 函数和ExceptionCaught函数分别在处理完请求和捕捉到异常时被调用。HttpServerHandler的完成一次请求处理的过程如流程图2.2所示。

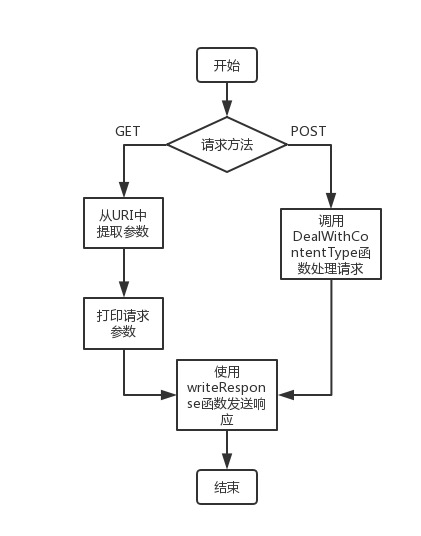


图2.2 HttpServerHandler处理流程

从图2.2可以看出，对于GET请求的处理比较简单，实际上，只需要在程序中初始化一个queryDecoder实例完成GET请求的解析，对接收到的请求只是直接打印参数，可以根据实际的需要完成更复杂的功能。dealWithContentType函数和writeResponse函数分别完成POST请求的处理和响应的发送。dealWithContentType对请求的处理流程如图2.3所示。

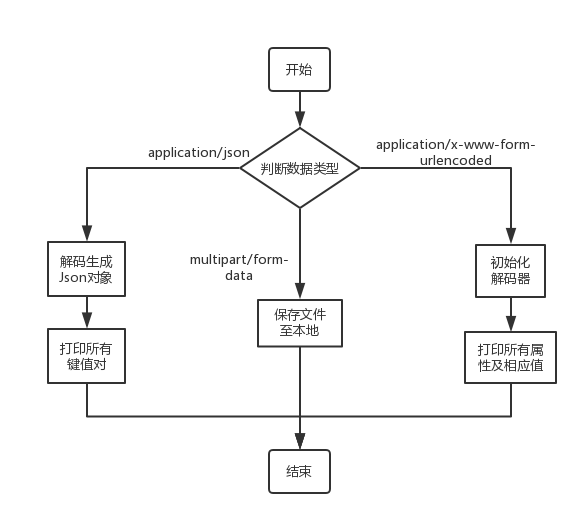


图2.3 dealWithContentType函数处理请求流程

服务器端成功接受请求后调用writeResponse函数发送响应给客户端，实验程序中正常情况下发送消息”success”，异常情况下发送”error”。writeResponse函数设置响应状态码，根据是否要断开连接选择是否要将消息写入响应和断开连接。writeResponse函数处理消息的流程如图2.4所示。

断开连接为真条件：请求的首部CONNECTION字段为"close"，或者协议版本为HTTP1.0、CONNECTION字段不为"keep-alive"，或者服务器端设置forceClose=true。

正常发送响应的过程会将状态码设置为HttpResponseStatus.OK（200），如果在处理请求过程中如果捕获异常，在catch语句块中会设置状态码为HttpResponseStatus.InternalServerError（500），并设置函数参数forceClose为true。

如果要断开连接，则发送一个内容为空的响应报文。

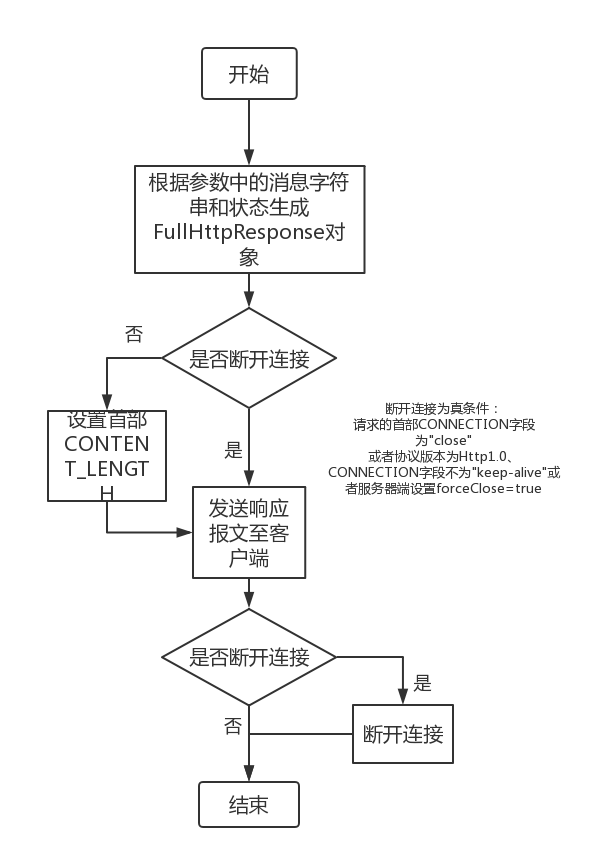


图2.4 writeResponse函数发送响应流程

客户端程序HttpClient用来测试服务器的功能。HttpClient中实现了三个函数：sendGet()以GET方法发送请求、sendPostJson以POST方法发送请求，Content-Type为application/json、sendPostForm以POST方法发送请求，Content-Type为application/x-www-form-urlencoded。HttpClient持续接收标准输入以选择调用哪一个函数。每次发送消息的过程如图2.5所示。

发送、接收消息分别由sendMessage和receiveMessage函数完成。sendMessage根据发送方法、发送内容类型确定发送的消息的首部内容并发送发消息，receiveMessage函数根据响应报文的状态显示相应信息。

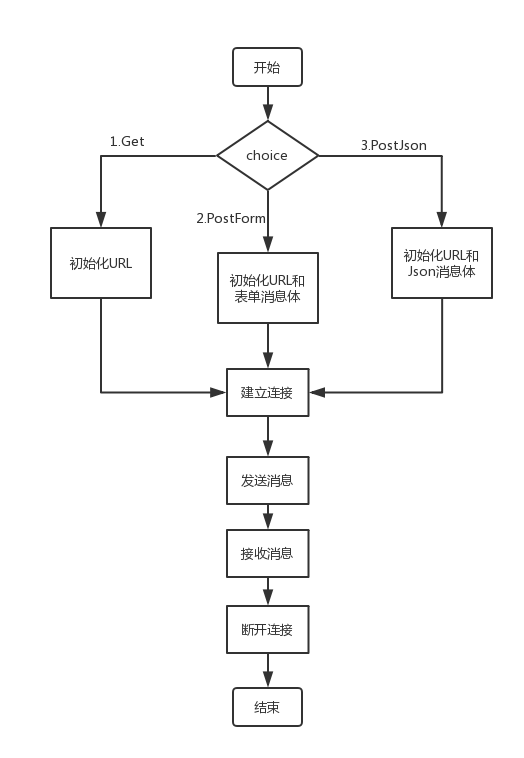


图2.5 HttpClient发送、接收消息流程图

## 实验过程

开发语言：Java

开发环境：IntelliJ IDEA

参考资料：《Netty权威指南》、《HTTP权威指南》、[netty5 HTTP协议栈浅析与实践](https://www.cnblogs.com/cyfonly/p/5616493.html)

实验使用Maven进行项目管理，依次编写服务端程序实现HTTP请求处理以及生成响应，编写客户端程序检验服务端是否能够正常收发消息。

## 实验结果及分析

**4.1实验结果**

运行实验程序时，先运行服务器端程序HttpServer。HttpServer正常启动运行结果如图4.1所示。

../../Screen%20Shot%202018-06-17%20at%2011.01.01%20AM.png

图4.1 HttpServer启动运行结果

HttpServer启动后，再运行客户端程序HttpClient。HttpClient运行结果如图4.2所示。为了检验不同请求方式及Content-type是否都能够处理，依次选择三种不同方式发送请求。

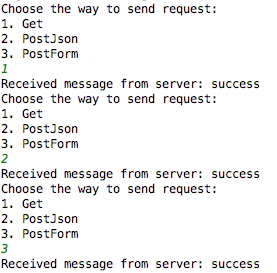


图4.2 客户端程序HttpClient运行结果

服务器端接收消息结果分别如图4.3、4.4所示。

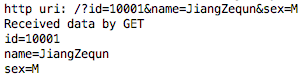


图4.3 服务器端接收、处理GET请求

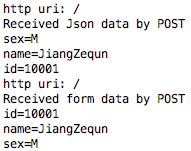


图4.4服务器端接收、处理不同Content-type的POST 请求

**4.2实验结果分析**

从实验结果中，可以看出HTTP协议的POST、GET方法基本实现。GET方法对URL进行编码，在uri中包含了请求参数，而POST方法没有，两者的差别显示在服务端运行结果中。